长三角先进材料研究院

Yangtze Delta Region Institute of Advanced Material

实习项目报告

数据库管理系统

前端技术实现：卢光帅 后端技术实现：孟世元

作者：孟世元

主管老师：赵海龙

时间：2022年6月26日至9月15日

目录

第一部分：技术使用

前端使用的技术为 VUE 2.0+Element UI。

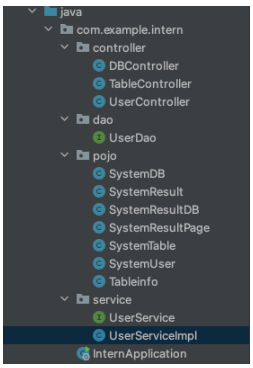
VUE 是一个用于创建用户界面的开源 JavaScript 框架，也是一个创建单页应用的 Web 应用框架；Vue 所关注的核心是 MVC 模式中的视图层，同时，它也能方便地获取数据更新，并通过组件内部特定的方法实现视图与模型的交互。

Element ui 它是由饿了么前端团队推出的基于 Vue 封装的 UI 组件库，提供 PC 端组件，简化了常用组件的封装，降低开发难度。简单的来说，通过使用 element ui 可以直接使用一些别人设计好的界面布局与样式。网址为：https://element.eleme.cn/#/zh-CN。下图为网站内部的展示。

图一：element UI 中对于按钮的各种样式

后端使用的技术为 Springboot(2.7.3) + JAVA（jdk8） + Mybatis + MySQL

SpringBoot 是所有基于 Spring 开发的项目的起点。Spring Boot 的设计是为了让你尽可能快的跑起来 Spring 应用程序并且尽可能减少你的配置文件。简单来说就是 SpringBoot 其实不是什么新的框架，它默认配置了很多框架的使用方式，就像 maven 整合了所有的 jar 包，spring boot 整合了所有的框架（不知道这样比喻是否合适）。在 Springboot 中，我引入了四个层，分别为 controller，dao，service，pojo。



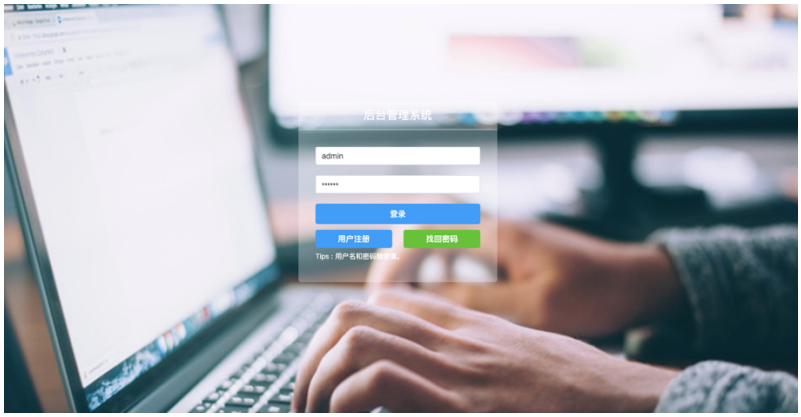
图二：本项目中的四个层级

MyBatis 是一个开源、轻量级的数据持久化框架，是 JDBC 和 Hibernate 的替代方案。MyBatis 内部封装了 JDBC，简化了加载驱动、创建连接、创建 statement 等繁杂的过程，开发者只需要关注 SQL 语句本身。

第二部分：功能与界面

界面一：登陆与注册界面

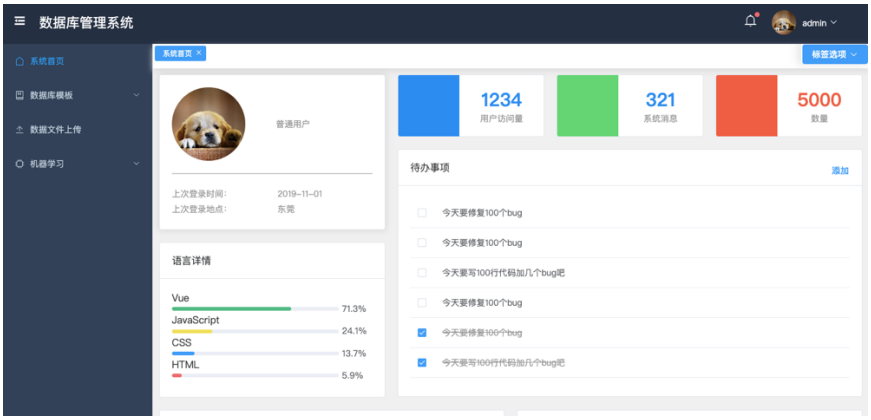
功能介绍：用户可以在本界面使用个人的账号与密码进行登录，如果没有可以通过点击注册按钮进行注册。注册成功后即可登录。



图三：登陆与注册界面

界面二：系统首页

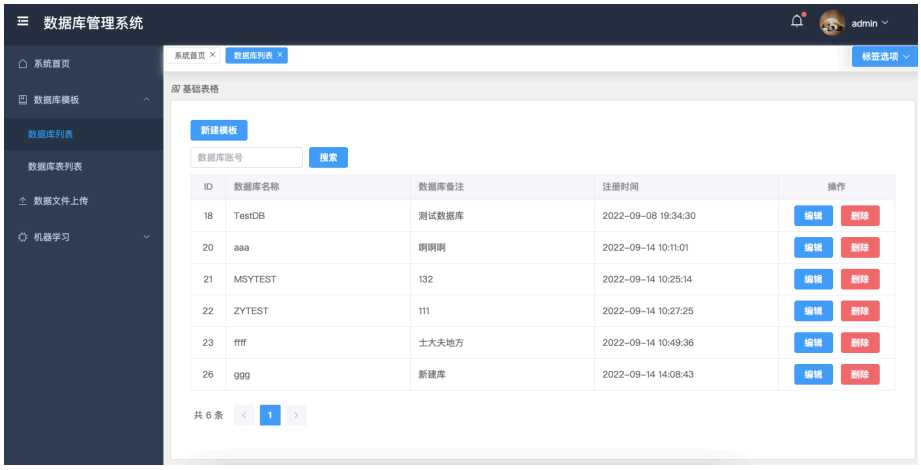
功能介绍：用户登陆成功后将跳转至本页面，通过点击右上角的头像可以进行退出登录这项操作。同时，左侧的分栏可以使用户清晰的看到本网站所提供各项功能。



图四：用户主界面

界面三：数据库界面

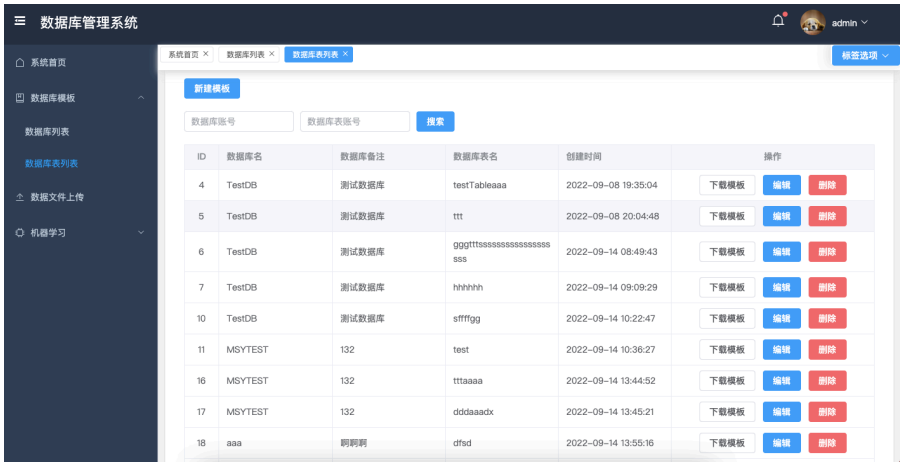
功能介绍：1. 用户在本界面可以看到自己名下有哪些数据库，下图中以用户 admin 为例。同时各个数据库的相关信息如数据库名称，数据库备注，创建时间均可看到。2. 用户可以通过点击“新建模板”并输入数据库名与备注名在自己名下新建一个数据库。3. 用户可以在搜索框内输入部分信息，查询有关的数据库。（模糊查找：如仅输入 f 便可以找到 ffff 这个数据库。）4. 通过点击“编辑”按钮，编辑数据库备注名。5. 通过点击“删除”按钮，删除对应的数据库。6. 分页展示功能，为防止数据库过多，在本页面最多一次性只可以显示 10 个数据库。



图五：数据库相关界面

界面四：数据库表界面

功能介绍：1. 以 admin 用户为例，用户可以看到在自己名下有哪些数据库表。同时还有与各张表相关的信息，如表属于哪个数据库，表的名字，表的创建时间。2. 用户可以通过点击“新建模板”按钮，并选择相应的数据库，输入要创建的表的名称，以及表的各个字段名称以及数据类型进行新表的创建。3. 提供了两个搜索框，左侧搜索框可以根据用户输入的数据库相关信息进行搜索，右侧搜索框可以根据用户输入的数据库表的相关信息进行搜索。左右两侧搜索框同时输入可以进行数据库与数据库表的联合查询。4. 用户可以通过点击“编辑”按钮进行对某张表名字的修改，同时可以通过输入相关信息在这张表中新建一些字段。5. 用户可以通过点击“删除”按钮对对应的数据库表进行删除。6. 用户可以通过点击“下载模板”按钮，根据数据库表的字段信息下载一张带有相关信息的 excel 表格。



图六：数据库表的相关界面

第三部分：后端功能实现细节

一：四个层级的用处

本项目中一共分了四层。controller：控制层。pojo：实体层。dao：数据层。service：业务逻辑层。

这四层的关系：

(1)实体类是属性对象，用于供给其他层引用，该层的属性往往和数据库的表结构各个字段相同（可以在 MySQL 或者 navicat 中使用 desc 表名 查看表的结构）。

(2)dao 层往往要写上@mapper 注解，告诉 springboot 这个是 mapper 接口。mapper 层所定义的接口要实现对数据的操作可以采用两种方式：一个是加上@mapper 注解之后，采用 sql 语句。另一种是继承 mapper 接口（extends mapper()<>）。

(3)service 层负责功能的编写。将所需要的功能都定义在一个 service 层的接口当中，再创建一个包，该包中定义实现类，用来实现方法，编写功能实现的代码。

(4)controller 往往定义一个 service 接口的对象，然后调用里面的方法。接着将结果输出即可。一般要加@RestController，该注解的作用将结果以 json 的格式返回。RequestMapping 用来和 http 请求进行交互。将 http 所相应的请求添加到该 Rest 控制器里面的方法中。在 controller 层定义 service 对象的时候要加@Autowired 自动注入。

二：以新建数据库这项功能为例，描述项目功能开发的流程。

第一步：构思新建数据库这项功能所需要的 SQL 语句，并将 SQL 语句填入 mapper.xml 这个文件中。（为防止后续 sql 语句过多，在每个新写入的 SQL 语句之上都要加入相关含义的注释）。

```
<!--第一个界面，与数据库表有关的相关操作-->
<!--使用所需要的数据库-->
<update id="useDB">
    use ${dbname};
</update>

<!--新建一个库-->
<update id="creatDB" parameterType="String">
    CREATE DATABASE ${dbname};
</update>
<!--检测数据库是否存在-->
<select id="DBExist" resultType="com.example.intern.pojo.SystemDB">
    SELECT * from intern.SystemDB where dbname=#{dbname};
</select>
<!--向数据库表中写入创建的数据库的信息-->
<insert id="updateDB">
    INSERT into intern.SystemDB(`dbname`,`editor`,`edittime`,`note`) values (#{dbname},#{editor},#{edittime}, #{note});
</insert>
```

图七：与新建数据库这项功能有关的 SQL 语句

第二步：在 dao 层中根据 mapper.xml 文件中所提供的 SQL 语句完成 JAVA 接口的设计。

```
//使用特定的数据库
11 usages
void useDB(@Param("dbname")String dbname);
//用户创建数据库
1 usage
void creatDB(@Param("dbname")String dbname);
//检测数据库是否存在
1 usage
SystemDB DBExist(@Param("dbname")String dbname);
//将新数据库的信息写入数据库表中
1 usage
void updateDB(@Param("dbname")String dbname, @Param("note")String note, @Param("edittime")String edittime, @Param("editor")Integer editor);
```

图八：dao 层中接口的书写，应与 mapper.xml 文件中的 SQL 语句对应

第三步：在 pojo 层中定义两个对象。SystemDB（数据库对象）与 SystemResult（从后端向前端返回的数据及各式对象）。

```
package com.example.intern.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

53 usages
@Data
@AllArgsConstructor
@NoArgsConstructor
public class SystemResult<T> {
    //给用户返回的消息
    private String msg;
    //数据是否正常请求
    private Integer code;
    //具体返回的数据
    private T data;
}
```

```
package com.example.intern.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

18 usages
@Data
@AllArgsConstructor
@NoArgsConstructor
public class SystemDB {
    public String dbname;
    public Integer id;
    public Integer editor;
    public String edittime;
    public String note;
}
```

图九与十：pojo 层中两个对象的具体建立

第四步：先在 service 层中的 UserService.java 文件中写入“新建数据库”这项功能的接口，并后续在 UserServiceImpl.java 中写“新建数据库”这项功能的具体实现逻辑。

具体逻辑为：1. 通过用户输入的数据库名，在后端一张名为“SystemDB”的表中进行查询，查看是否有与用户输入的同名库，如果存在则不能完成创建，请用户重新输入并创建。2. 如果没有同名数据库，首先记录当前用户操作的时间，并记为数据库的创建时间。3. 将用户输入的数据库名，数据库备注名，以及生成的数据库创建时间传入“SystemDB”这张表中。4. 调用相关的 SQL 语句，根据用户传入的数据库名完成对数据库的创建。5. 最后，将数据库的各项信息传入一个新建的 SystemDB 对象之中，并在 SystemResult 中的 data 部分进行引用。将 SystemResult 中的 msg 设为“创建成功”，将 SystemResult 中的 code 设为 1（0 为失败，1 为成功）。

```java
//用户创建新的数据库
1 usage   1 implementation
public SystemResult creatDB(String dbname, String note, Integer editor);
```

图十一：UserService.java 中用户创建新的数据库这项功能的接口

```java
//用户新建一个数据库
1 usage
@Override
public SystemResult creatDB(String dbname, String note, Integer editor) {
    SystemResult result = new SystemResult();
    result.setCode(0);
    result.setData(null);
    try{
        //检测数据库之前知否存在
        SystemDB existDB = userDao.DBExist(dbname);
        if(existDB != null){
            //如果数据库已存在
            result.setMsg("数据库已存在,请输入新的数据库名");
        }else{

            Date date = new Date();
            SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd :HH:mm:ss");

            //将新的数据库信息写入数据库表
            userDao.updateDB(dbname, note, dateFormat.format(date), editor);
            //创建新的数据库
            userDao.creatDB(dbname);
            result.setMsg("数据库创建成功");
            result.setCode(1);
```

图十二：UserServiceImpl.java 中功能的具体实现（part1）

```
            SystemDB DB = new SystemDB();
            DB.setDbname(dbname);
            DB.setNote(note);
            DB.setEdittime(dateFormat.format(date));
            DB.setEditor(editor);
            result.setData(DB);
        }
    }catch (Exception e) {
        result.setMsg(e.getMessage());
        e.printStackTrace();
    }
    return result;
}
```

图十三：UserServiceImpl.java 中功能的具体实现（part2）

第五步：在 Controller 中完成前后端交互的接口。

```
//用户创建新的数据库
@PostMapping(value = "/creatDB")
public SystemResult creatDB(@RequestBody JSONObject a, HttpServletRequest request){
    //从前端获取三个相关的参数
    String dbname =a.getString( key: "dbname");
    String note = a.getString( key: "note");
    Integer editor = Integer.valueOf(request.getHeader( name: "token"));
    return userService.creatDB(dbname, note,editor);
}
```

图十四：Controller 的 java 具体代码

第四部分：个人在使用 springboot 即各项技术时所遇到的困难。

创建项目时 start.spring.io 连接超时

进入到 IDEA 的 preference 中搜索 HTTP Proxy，选择 Auto-detect proxy settings

点击最下面的 Check connection 弹出如下的输入框，输入地址 https://start.spring.io 点击 ok，如果 successful 证明连接成功，（如果 successful 没有出现，多尝试几次只要通一次就可以了，表明网站可以连通）

前后端交互时 Form data 与 JSON data 的区别

一般都是在文件上传的时候使用 form 表单，大部分前后端交互还是使用 JSON data。

所以需要使用@RequestBody 将前端传过来的数据解析出来。

```
//用户登陆
@PostMapping(value = "/login")
public SystemResult login(@RequestBody JSONObject a){
    String username = a.getString( key: "username");  //从前端获取用户名
    String password = a.getString( key: "password");
    return userService.login(username, password);}
```

前后端 url 设置—中间可以再加一段 api，使得项目结构更加合理。与跨域问题相关。

如何获取前端 http 请求头中所带的信息

HttpServletRequest request 作为函数的参数传入方法，之后在方法内部调用。

```
Integer editor = Integer.valueOf(request.getHeader("token"));
```

java.lang.NumberFormatException: For input string: "undefined"问题

这个问题是当时在添加分页功能的时候出现的，具体问题在解决了分页问题传入的参数后便消失了。

跨域问题

直接在 handler 方法上加一个@CrossOrigin 注解即可。还有另外别的方法可以去网上搜。

SQL 语法问题—检查 dao 层文件与 mapper.xml 文件中方法名字的拼写

Invalid bound statement (not found):
com.example.intern.dao.UserDao.searchDBthrougnDBname